Celoxica

Platform Developer's Kit

SimulinkDSM Manual

Celoxica, the Celoxica logo and Handel-C are trademarks of Celoxica Limited.

All other products or services mentioned herein may be trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous development and improvement. All particulars of the product and its use contained in this document are given by Celoxica Limited in good faith. However, all warranties implied or express, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Celoxica Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

The information contained herein is subject to change without notice and is for general guidance only.

Copyright © 2005 Celoxica Limited. All rights reserved.

Authors: RG

Document number: 1

Customer Support at http://www.celoxica.com/support/

Celoxica in Europe	Celoxica in Japan	Celoxica in the Americas
T: +44 (0) 1235 863 656	T: +81 (0) 45 331 0218	T: +1 800 570 7004
E: sales.emea@celoxica.com	E: sales.japan@celoxica.com	E: sales.america@celoxica.com

Contents

1 DSM SIMULINK 4

2 INDEX 13

Conventions

A number of conventions are used in this document. These conventions are detailed below.

Warning Message. These messages warn you that actions may damage your hardware.

Handy Note. These messages draw your attention to crucial pieces of information.

Hexadecimal numbers will appear throughout this document. The convention used is that of prefixing the number with '0x' in common with standard C syntax.

Sections of code or commands that you must type are given in typewriter font like this:

```
void main();
```

Information about a type of object you must specify is given in italics like this:

```
copy SourceFileName DestinationFileName
```

Optional elements are enclosed in square brackets like this:

```
struct [type_Name]
```

Curly brackets around an element show that it is optional but it may be repeated any number of times.

```
string ::= "{character}"
```

Assumptions & Omissions

This manual assumes that you:

- have used Handel-C or have the Handel-C Language Reference Manual
- are familiar with common programming terms (e.g. functions)
- are familiar with MS Windows

This manual does not include:

- instruction in VHDL or Verilog
- instruction in the use of place and route tools
- tutorial example programs. These are provided in the Handel-C User Manual

1 DSM Simulink

This manual documents the suggested use of DK and Handel-C with Matlab and Simulink.

1.1 Matlab & Simulink Overview

Matlab is a design environment and language from The MathWorks™. The aim of the Matlab language is to provide a high-level environment for modelling algorithms. It bears superficial similarities to the C language in its use of operators and imperative style but has built in support for matrix and array processing.

It also differs in its relaxed approach to memory management; variables are dynamically created as they are needed. Extensive use is made of in-built functions that allow the user to manipulate matrices and common mathematical functions such as matrix inversion, or generating geometric series.

It is widely used within the scientific and engineering community to model complex algorithms – before implementing the algorithms in a "real world" system.

As embedded systems become more complex (and algorithmic) the necessity arises to be able to model the algorithms fully before implementing them. With this rise in complexity more embedded designs are modelled first in Matlab (and/or Simulink) before their subsequent implementation. As a result, the need to be able to push the implementation cycle through quickly has arisen.

Since DK is a high-level system design tool that relies heavily on software-style simulation and implements complex algorithmic designs it is natural to provide a route from Matlab to a hardware implementation. We focus here on the ability to verify hardware designs written in Handel-C (amongst other languages) in the Matlab/Simulink environment.

The process of porting M-code to Handel-C is not covered here, as it is similar to that of porting C code to Handel-C.

1.2 Creating Object files for Matlab

Matlab MexFunctions:

Since DK allows you to write Handel-C code that can compiled into an object file, you can use this object file (and the functionality encapsulated within it) as the basis of the mexFunction. We can then call this mexFunction just as we would any other Matlab call.

The way to export Handel-C functions is to use `extern "C"`. Using `extern "C"` in a Handel-C file simply alters the linkage for the functions listed within the braces. They are

exported and made available to other (top-level) C programs. This means you can still write a normal program in Handel-C and use any functions in both your Handel-C file, or a C-file.

Within the PDK install find the timestwo.c file under CELOXICA_PDK_HOME\Examples\CoSim\Matlab. We are going to use this as the basis for an example showing how to call Handel-C from the matlab cmd line. The part of the file that modifies the data is implemented in a function called `timestwo`. We are going to replace this function (written in C) with one written in Handel-C.

Creating a New mexFunction Project

It is a good idea to set up a new active configuration in DK that handles the compilation to an object file. (Creating Build Configurations in DK help) Use this backend simulator cmd line in the new configuration:

cl -c /Oityb1 /I"CELOXICA_DK_HOME\sim\include" "%1" /Fo%3.obj

Copy and paste the following code into a new project.

```
set clock = external;
extern "C"
{
   void timesTwoKernel( int 32 x, int 32 *y)
   {
      *y =  x * 2;
   }
}

void main(void)
{
   int a, b;

   a = 34;
   b = 0;

   delay;
   timesTwoKernel( a, &b);
   delay;
}
```

It is also important to include a `set clock =` statement. Whilst the code will compile without this statement, the function will not have a clock associated with it, and as a result will output incorrect results.

Once you have compiled the code, it is possible to use the resulting .obj file as you would any other C static library file. By default, the output object file will have the same name as your project.

Creating the mexFunction

In order to use the Handel-C function in the C program, we modify the timestwo.c file slightly. This essentially becomes a wrapper file, which has the necessary function calls to interface directly to the matlab mex API. The wrapper gets the data from the API and passes it to the Handel-C kernel. Notice that, since we have specified that the Handel-C function uses `int 32` argument types, the calling procedure must explicitly pass in 'C' integer type `int`. The function prototype will look like this:

```
void timesTwoKernel( int x_int, int *y_int);
```

The timestwo.c calling procedure now looks like this:

```
int x_int, y_int;
double x, y;
/* Assign pointers to each input and output. */
  x = mxGetPr(prhs[0]);
  y = mxGetPr(plhs[0]);
  /* Call the timestwo subroutine. */
  x_int = tofixed( x[0] );
  timesTwoKernel( x_int, &y_int);
  *y = todouble( y_int );
```

In Matlab, the default data type is double. In Handel-C floating point representations are not in-built types, so we must alter any double to an integer, scaling before passing to the Handel-C kernel. Here we use a simple scaling, wrapped up in a macro. Using scaling in this manner does not automatically reduce accuracy, although if further accuracy is needed, it is possible to use the HCNum wide number library.

Follow the procedures associated with your chosen C/C++ compiler to compile the timestwo.c wrapper and link in the Handel-C generated kernel to produce a Dynamic Link Library (DLL). Matlab requires you to export a symbol called `mexFunction`.

A batch file has been written to help Visual C++ users to create mexfunctions easily. It is called createmexFunction.bat and resides in CELOXICA_PDK_HOME\software\bin directory. It requires two arguments. The first is the name of the "C" wrapper file (e.g. timestwo.c) and the second is the name of the object file kernel generated from Handel-C.

The DLL can be quickly tested from the Matlab cmd line. However, it is necessary that the DLL be either in Matlab's current directory or in its list of paths. For more details on paths in Matlab, consult the MathWorks documentation. To test the DLL use the following code:

```
>> x = 3;
>> y = times2(x)
y =
     6
```

1.3 Using Simulink with DK

Within the Simulink environment there is the ability to write user defined functions similar to mexFunctions. These are called S-Functions. The API is similar to that of the mexFunction, but has been extended to support the cycle-based simulation that Simulink is based on. For more information about S-Functions, including full documentation on how to write them, refer to the MathWorks documentation.

It is possible create kernel functions in Handel-C and integrate them into an S-function, using a similar procedure to that described in Creating Object files for Matlab. This process is described in detail in Co-simulation support for MATLAB

1.4 Simulink & DSM

Using the capability of DSM to connect a software environment to DK or “real hardware” it is possible to connect an S-Function and DK using DSM. Since DSM is implemented for simulation as well as various hardware platforms, it is possible to both co-simulate and also to drive real hardware designs from Simulink.

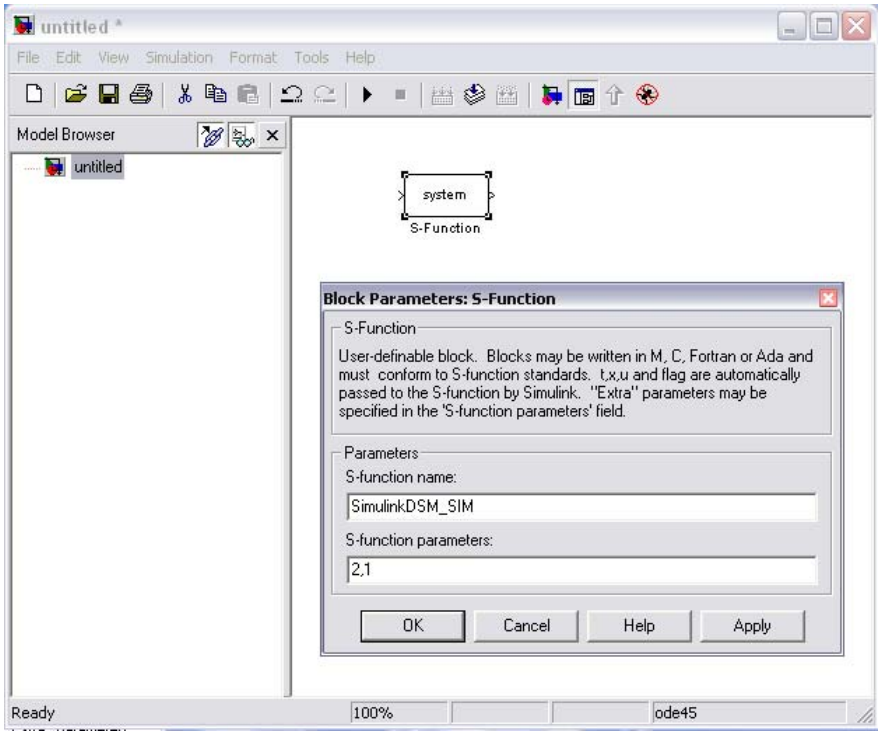
Simulink and DK in Co-Simulation Via DSM

The DSM co-simulation S-Function is provided as a Dynamic Link Library (DLL). In order to use the DLL in a co-simulation, it is necessary to set the path to the DLL in Matlab. Open Matlab and go to File | Set PathSelect Add Folder and navigate to CELOXICA_PDK_HOME\Software\Bin. Save the settings. Alternatively, use the addpath function from the Matlab cmd line.

```
>> addpath('c:\PDK\software\bin');
```

Creating a new Co-Simulation Project

In order to use the DLL in an S-Function, drag and drop an S-Function token from the list of library block sets in the Simulink Library Browser into your Simulink project. Set S-Function Name to the name of the DLL (SimulinkDSM_SIM). In S-Function Parameters specify the number of S2H ports followed by the number of H2S ports, separated by a comma. For instance for 2 S2H ports and 1 H2S port put “2,1”.



S-FUNCTION PARAMETERS FOR SIMULATION

Any DSM design could be co-simulated with Simulink. However, the design must have matching ports (as any valid DSM design must have).

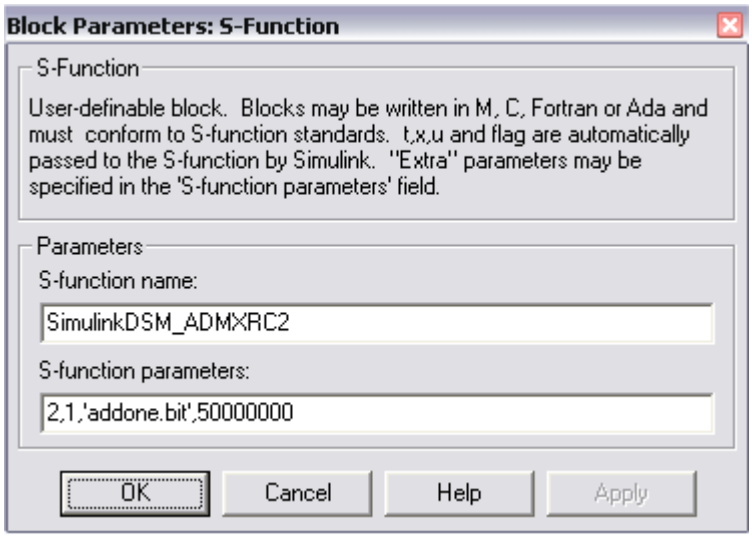
Using Simulink with an FPGA Design

As well as co-simulation between DK and Simulink, there is also the facility to connect Simulink to a design running on an FPGA. The RC2000/ADMXRC2 platform from Celoxica is a PCI board supported by DSM. If the board is resident within a host PC, it is possible to communicate with the FPGA in real-time using DSM.

Substitute the SimulinkDSM_SIM DLL for the SimulinkDSM_ADMXRC2 DLL in the design above. As well as specifying the number of S2H and H2S ports for your design, it is also necessary to specify a string locating the configuration bitfile for the FPGA and a clock frequency. This can be a relative or an absolute path to the bitfile and must include the file suffix. For instance if your bitfile is the “Addone” example from DSM examples use

'C:\PDK\Examples\DSM\AddOne\ADMXRC2\addone.bit'. Simulink expects single quote marks to be used.

The clock frequency is passed to the programmable clock generator on the RC2000. It should match the value specified in the Handel-C project. This must be passed as an integer. For instance, to run the FPGA at 40MHz, use 40000000.


Block Parameters: S-Function
S-Function
User-definable block. Blocks may be written in M, C, Fortran or Ada and must conform to S-function standards. t,x,u and flag are automatically passed to the S-function by Simulink. "Extra" parameters may be specified in the 'S-function parameters' field.
Parameters
S-function name:
SimulinkDSM_ADMXRC2
S-function parameters:
2,1,'addone.bit',50000000
OK
Cancel
Help
Apply

S-FUNCTION PARAMETERS FOR RC2000

Choosing Start (Ctrl + T) from Simulink will run the design in hardware. The results will be passed back to a scope in an identical manner to when running in simulation.

Explanation

For an in-depth explanation of how Simulink works please consult the MathWorks Documentation.

In Simulink each block in the flowgraph is executed in order as defined by its connections. A simulation cycle encompasses the longest path period for data token passing from a source to a sink. When progress of a simulation cycle reaches the S-Function, the `mdlOutputs` callback is executed. This oversees the subsequent sending and receiving of data. Firstly, the value on each of the input ports is sampled. The data flow passes into DK domain. The end of the data processing on the hardware/DK side is signalled by the writing of a DsmWord from the hardware side. The data flow returns to Simulink and the simulation cycle continues. State information about the progress of the simulation cannot explicitly be sent to the DK/hardware side.

1.5 DSM Simulink Example

LMS Filter Example

The co-simulation example is available in the directory CELOXICA_PDK_HOME\PSL\Cosim\SimulinkDSM.

To run the example, you must have DK3.0 or greater, the Platform Developers Kit, MATLAB6p1 and Simulink v4.0 installed. The example demonstrates an LMS filter, written in Handel-C being co-simulated with Simulink.

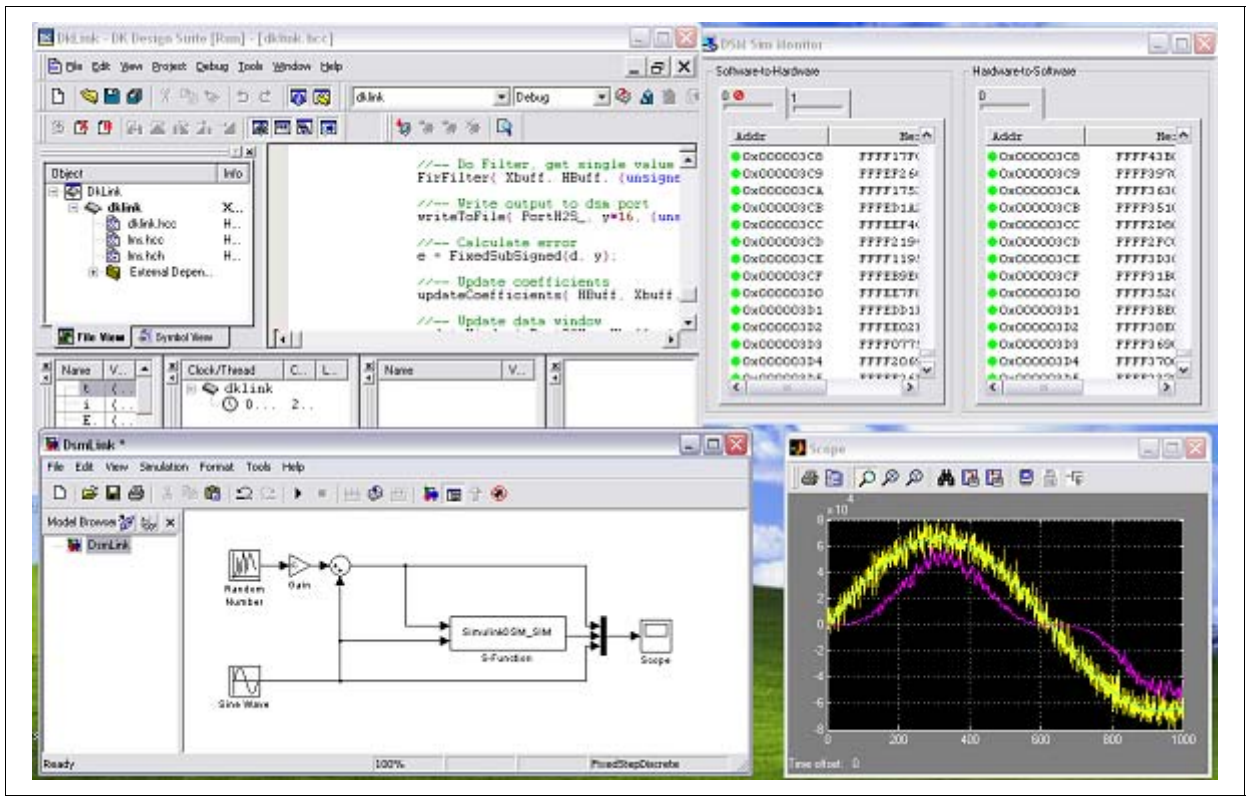
Study the LMS filter example. It uses a simple model to clean a large amount of noise added to a sine-wave. The model assumes that for every Simulink simulation cycle, a noisy signal and a desired signal are made available to the LMS core. The LMS core uses the desired signal characteristics to clean the noisy signal input. For more information on the behaviour of LMS filters consult a good book on DSP. (eg Digital Signal Processing – A Practical Approach – E.C. Ifeachor, B.W.Jervis)

Running the Example

It is first necessary to ensure that the path to any dynamic linked libraries that Simulink S-Functions might use are set. Open Matlab and go to File | Set Path. Select Add Folder and navigate to CELOXICA_PDK_HOME\Software\Bin. Save the settings. Then

1. Open the DK workspace. Rebuild the DK side (F7)
2. Open the Simulink (.mdl) workspace
3. Starting the simulation from the DK side (F5) and then the Simulink side (Ctrl + T).

It is possible to insert breakpoints and debug the Handel-C code just as you normally would in a DK simulation. If you choose to do this, it may appear that as the simulation pauses, Simulink appears to freeze. This is because Simulink is single-threaded process that is waiting for a return data value from DSM. As the simulation cycle finishes, it will refresh again.Using the DSMSim monitor It is also possible to the view the data passed between Simulink and Handel-C simulators.



SCREENSHOT: DSM SIM AND SIMULINK

Note:The LMS core code is not optimized and is taken almost directly from C that was initially used to model it. Modifications include adding fixed point support and using DSM as the I/O basis, instead of files. It is left as an exercise for the reader to optimize the code. It should be possible to optimize the core so that a result is output on every DK simulation clock cycle.

2 Index

L

LMS Filter..... 10

S

SimulinkDSM..... 4